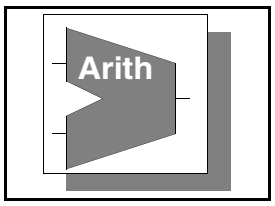
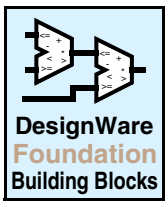




DW_log2

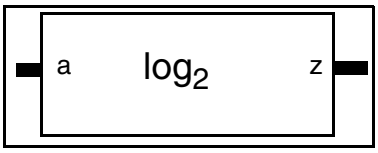
Base-2 Logarithm

Version, STAR, and myDesignWare Subscriptions: IP Directory

Features and Benefits

- Parameterized word width
- Supports up to 60 bits of precision

Revision History



Description

DW_log2 computes the base-2 logarithm ($\log_2$) of an input a in fixed-point format. The input must be in the range (1,2) (normalized input) and therefore the output is in the range (0,1).

The number of bits used as input and output is defined by a parameter (*op_width*). The input has 1 integer bit and *op_width* - 1 fractional bits. The output has *op_width* fractional bits.

The component implements the logarithm function using different algorithms depending on the number of input bits. The selection of the algorithm is done automatically to deliver the best QoR.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
a	*op_width* bits	Input	Input data in the range (1,2).
z	*op_width* bits	Output	$\log_2$(a) in the range (0,1)

Table 1-2 Parameter Description

Parameter	Values	Description
op_width[a]	2 to 60	Word length of a and z
arch	0 to 2 Default: 2	Implementation selection ■ 0 = Area optimized ■ 1 = Speed optimized ■ 2 = 2007.12 implementation
err_range	1 or 2 Default: 1	Error range of the result compared to the infinitely precise result ■ 1 = 1 ulp ■ 2 = 2 ulps

a. The synthesis model fully supports this range, as does the Verilog simulation model in VCS, but the VHDL simulation model (in all simulators) and the Verilog simulation model in non-VCS simulators are limited to a range of 2 - 38. For details, see "Simulation" on page 4.

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Implement using the Datapath Generator technology combined with static DesignWare components	DesignWare

Table 1-4 Simulation Model

Model	Function
DW04.DW_LOG2_CFG_SIM	Design unit name for VHDL simulation
dw/dw02/src/DW_log2_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_log2.v	Verilog simulation model source code

The truth table for the function implemented by this component when *op_width* = 4, *arch* = 2 and *err_range* = 1 is shown in Table 1-5. Notice that the MS bit of the input vector is expected to be 1, when it is not, the component output will deliver a meaningless value. Observe that the error is always less than the weight of the LS bit position of the output, in the example, less than 2^{-4}.

The *arch* parameter controls implementation alternatives for this component. Different values result in different numerical behavior. You should experiment with this parameter to find out which value provides the best QoR for your design constraints and technology. Using *arch* = 0 (area optimized implementation) usually provides the best QoR for most time constraints.

Another parameter, *err_range*, can be used to relax the error boundary and get an implementation with slightly better QoR. The error does not exceed 2 ulps when *err_range* = 2, and it does not exceed 1 ulp when *err_range* = 1. The maximum error is 2 ulps when *err_range* = 1 for the configurations with *op_width* values from 39 to 60 for *arch* 0 and *arch* 1, and with *op_width* values from 25 to 60 for *arch* 2. The error of 1 ulp corresponds to 2^{-op_width}.

Table 1-5 Truth Table (*op_width*=4, *arch*=2, *err_range*=1)

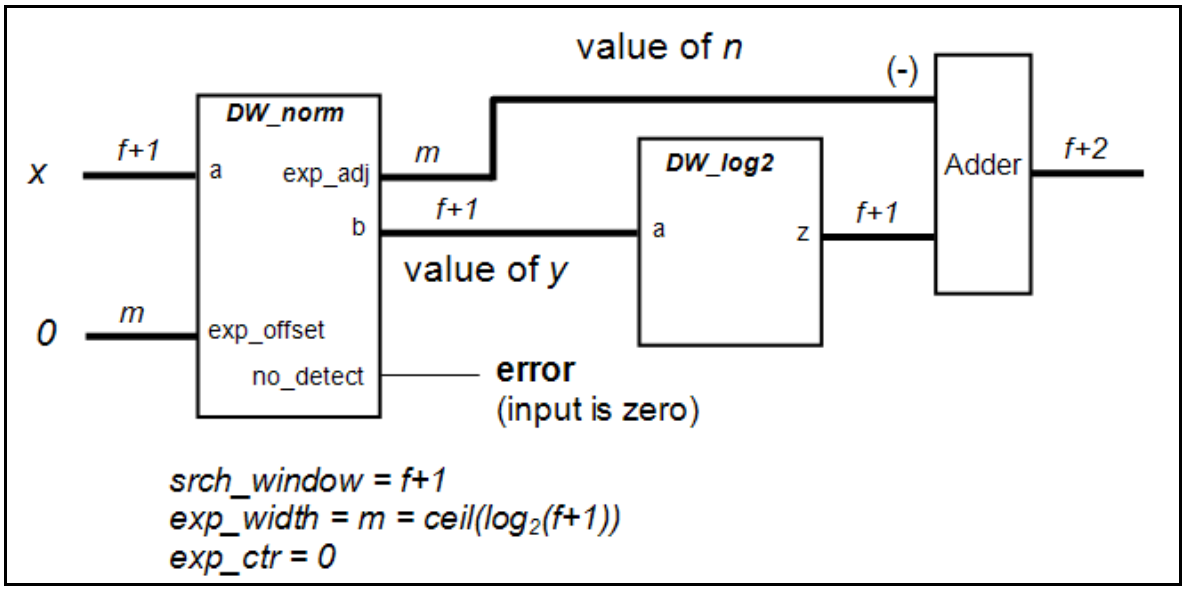
a(3:0)	a value (decimal)	z(3:0)	z value (decimal)	$\log_2$(a) (8 bits)
1.000	1.0	.0000	0.0	.00000000
1.001	1.125	.0011	0.188	.00101011
1.010	1.25	.0101	0.313	.01010010
1.011	1.375	.0111	0.438	.01110101
1.100	1.5	.1001	0.567	.10010101
1.101	1.625	.1011	0.688	.10110011
1.110	.175	.1101	0.813	.11001110
1.111	1.875	.1111	0.938	.11101000

The computation of the logarithm for other ranges of the input operand can be easily accomplished using some mathematical transformations. For example, to compute the $\log_2$ of a value x in the range (0,2) the designer may use the following identifies:

$$\log_2(x) = \log_2\left|\frac{x2}{2^n}\right| = \log_2\left|\frac{y}{2^n}\right| = \log_2(y) - n$$

where n corresponds to the number of zeros in MS bit positions. The value $y=2^n$ is obtained using a normalization unit (see DW_norm). The block diagram for the circuit that computes $log_2(x)$ for x in the range (0,2) is shown in Figure 1-1. Notice that x in the figure must be in the format x.xxxxxx (1 integer bit and f fractional bits).

Figure 1-1 Block Diagram of Circuit to Compute $\log_2(x)$, with x in the Range (0,2)



Consider the case when $x = 0.0110 = 3/8$. Using the proposed transformations we get $y = 1.100$ and $n = 2$. The base-2 logarithm of y is given as 0.1001 and thus 0.1001-(10.000) = 10.1001 which corresponds to the value -1.4375. The 8-bit value of $log_2(3/8)$ (excluding the sign bit) is obtained by other means as 10.1001011 (in two's complement).

Alternative Implementation of Base-2 Logarithm with DW_lp_multifunc

The base-2 logarithm operation can also be implemented by DW_lp_multifunc component, which evaluates the value of base-2 logarithm with 1 ulp error bound. There will be 1 ulp difference between the value from DW_lp_multifunc and the value from DW_log2. Performance and area of the synthesis results are different between the DW_log2 and base-2 logarithm implementation of the DW_lp_multifunc, depending on synthesis constraints, library cells and synthesis environments. By comparing performance and area between the base-2 logarithm implementation of DW_lp_multifunc and DW_log2 component, the DW_lp_multifunc provides more choices for the better synthesis results. Below is an example of the Verilog description for the base-2 logarithm of the DW_lp_multifunc. A new wire, `nc` needs to be declared to be used in this instantiation as the output of DW_lp_multifunc is wider. This wire can be left unconnected.

For more detailed information, see the DW_lp_multifunc datasheet.

```
wire [1:0] nc; // no connection
DW_lp_multifunc #(op_width, 32) U1 (
    .a({a, 1'b0}),
    .func(16'h0020),
    .z({nc, z}),
    .status(status)
);
```

Simulation

In most cases, the simulation model for DW_log2 delivers results that are bit-accurate with the synthesis model. Note the following details.

For Verilog simulations:

- When using VCS:
  - Results are bit-accurate with the synthesis model for all component configurations.
- When using simulators other than VCS:
  - If `op_width` <= 38, results are bit-accurate with the synthesis model
  - If `op_width` > 38, results are not bit-accurate with the synthesis model

    This scenario produces the following warning message:

```
*****************************************************
*  For op_width values greater than 38, the         *
*  simulation model of the DesignWare Library       *
*  Component DW_log2 is only bit accurate with its  *
*  synthesis model when using Synopsys VCS or       *
*  VCS-MX simulators.                               *
*  This instance has an op_width value of <value>.  *
*  For other simulators, this model delivers an     *
*  output that has the same error bounds as the     *
*  synthesis model, but not the same numerical value.*
*                                                   *
*  support@synopsys.com                             *
*                                                   *
*****************************************************
```

For VHDL simulations:

- When `op_width` <= 38, results are bit-accurate with the synthesis model
- When `op_width` > 38, results are not bit-accurate with the synthesis model

  This scenario produces the following warning message:

```
The DesignWare Library Component DW_log2 is not bit accurate with its synthesis model
when the parameter op_width is larger than 38. The Verilog simulation model running with
Synopsys VCS simulator is bit accurate with the synthesis model for the present value of
op_width.
```

Related Topics

- Application Specific – Data Integrity Overview
- DesignWare Building Block IP User Guide

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DW_Foundation_comp_arith.all;

entity DW_log2_inst is
      generic (
            inst_op_width : INTEGER := 8;
            inst_arch : INTEGER := 2;
            inst_err_range : INTEGER := 1
            );
      port (
            inst_a : in std_logic_vector(inst_op_width-1 downto 0);
            z_inst : out std_logic_vector(inst_op_width-1 downto 0)
            );
    end DW_log2_inst;


architecture inst of DW_log2_inst is

begin

    -- Instance of DW_log2
    U1 : DW_log2
        generic map (
                op_width => inst_op_width,
                arch => inst_arch,
                err_range => inst_err_range
                )
        port map (
                a => inst_a,
                z => z_inst
                );


end inst;

-- pragma translate_off
configuration DW_log2_inst_cfg_inst of DW_log2_inst is
for inst
end for; -- inst
end DW_log2_inst_cfg_inst;
```

```
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_log2_inst( inst_a, z_inst );

parameter inst_op_width = 8;
parameter inst_arch = 2;
parameter inst_err_range = 1;


input [inst_op_width-1 : 0] inst_a;
output [inst_op_width-1 : 0] z_inst;

    // Instance of DW_log2
    DW_log2 #(inst_op_width, inst_arch, inst_err_range) U1 (
                        .a(inst_a),
                        .z(z_inst) );

endmodule
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
June 2023	DWBB_202212.4	■ Removed note about ulp error range in Table 1-2 on page 1 and clarified ulp details just above Table 1-5 on page 2
December 2022	DWBB_202212.0	■ Added footnote to *op_width* parameter in Table 1-2 on page 1
September 2022	DWBB_202203.4	■ Added "Simulation" on page 4 to explain how results from the simulation model match the synthesis model
January 2020	DWBB_201912.1	■ Corrected port names for DW_lp_multifunc in "Alternative Implementation of Base-2 Logarithm with DW_lp_multifunc" on page 3
July 2019	DWBB_201903.3	■ Removed reference to minPower library in "Alternative Implementation of Base-2 Logarithm with DW_lp_multifunc" on page 3 ■ Added this Revision History table and the document links on this page

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
www.synopsys.com